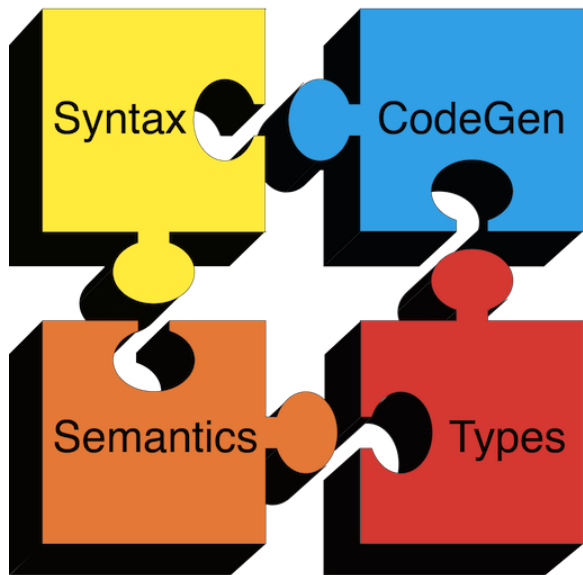


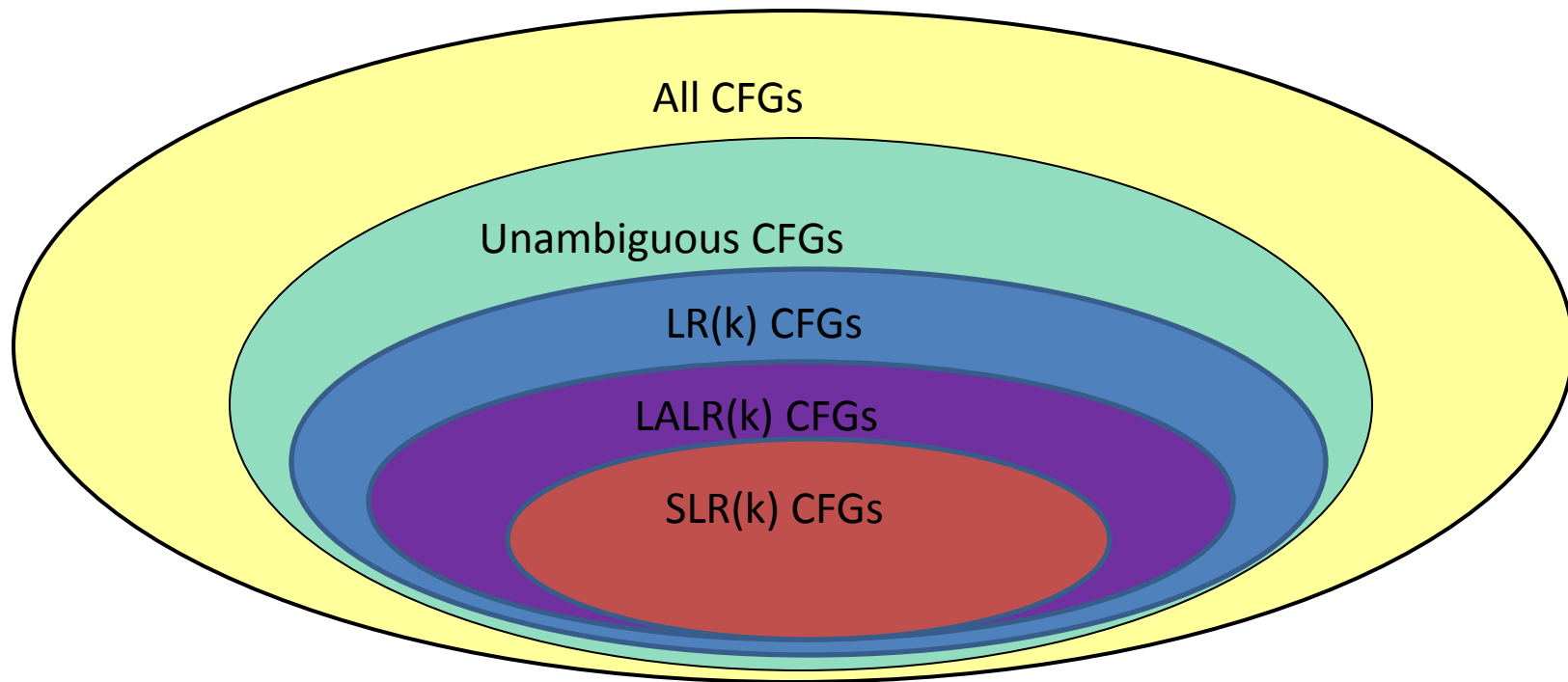


Compilers

Recognizing Handles

- **Bad News**
 - There are no known efficient algorithms to recognize handles
- **Good News**
 - There are good heuristics for guessing handles
 - On some CFGs, the heuristics always guess correctly

Recognizing Handles



- It is not obvious how to detect handles
- At each step the parser sees only the stack, not the entire input; start with that . . .

α is a viable prefix if there is an ω such that $\alpha | \omega$ is a state of a shift-reduce parser

- What does this mean? A few things:
 - A viable prefix does not extend past the right end of the handle
 - It's a viable prefix because it is a prefix of the handle
 - As long as a parser has viable prefixes on the stack no parsing error has been detected

Important Fact #3 about bottom-up parsing:

For any grammar, the set of viable prefixes is a regular language

- Important Fact #3 is non-obvious
- We show how to compute automata that accept viable prefixes

- An item is a production with a “.” somewhere on the rhs
- The items for $T \rightarrow (E)$ are
 - $T \rightarrow .(E)$
 - $T \rightarrow (.E)$
 - $T \rightarrow (E.)$
 - $T \rightarrow (E).$

- The only item for $X \rightarrow \varepsilon$ is $X \rightarrow \cdot$
- Items are often called “LR(0) items”

- The problem in recognizing viable prefixes is that the stack has only bits and pieces of the rhs of productions
 - If it had a complete rhs, we could reduce
- These bits and pieces are always *prefixes* of rhs of productions

Consider the input (int)

$E \rightarrow T + E \mid T$

$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$

- Then (E |) is a state of a shift-reduce parse
- (E is a prefix of the rhs of $T \rightarrow (E)$
 - Will be reduced after the next shift
- Item $T \rightarrow (E.)$ says that so far we have seen (E of this production and hope to see)

- The stack may have many prefixes of rhs's

$\text{Prefix}_1 \text{Prefix}_2 \dots \text{Prefix}_{n-1} \text{Prefix}_n$

- Let Prefix_i be a prefix of rhs of $X_i \rightarrow \alpha_i$
 - Prefix_i will eventually reduce to X_i
 - The missing part of α_{i-1} starts with X_i
 - i.e. there is a $X_{i-1} \rightarrow \text{Prefix}_{i-1} X_i \beta$ for some β
- Recursively, $\text{Prefix}_{k+1} \dots \text{Prefix}_n$ eventually reduces to the missing part of α_k

Consider the string $(\text{int} * \text{int})$:

$(\text{int} * | \text{int})$ is a state of a shift-reduce parse

$($ is a prefix of the rhs of $T \rightarrow (E)$

ϵ is a prefix of the rhs of $E \rightarrow T$

$\text{int} *$ is a prefix of the rhs of $T \rightarrow \text{int} * T$

The “stack of items”

$T \rightarrow (.E)$

$E \rightarrow .T$

$T \rightarrow \text{int} * .T$

Says

We’ve seen “(” of $T \rightarrow (E)$

We’ve seen ϵ of $E \rightarrow T$

We’ve seen $\text{int} *$ of $T \rightarrow \text{int} * T$

Idea: To recognize viable prefixes, we must

- Recognize a sequence of partial rhs's of productions, where
- Each partial rhs can eventually reduce to part of the missing suffix of its predecessor